REMARKS

Claims 11-54 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 103(a) Rejection:

The Office Action rejected claims 11, 13-19, 21, 23-28, 30-38, 40-46 and 48-54 under 35 U.S.C. § 103(a) as being unpatentable over Wang (U.S. Patent 6,292,936) in view of "The IR to VMx86 Translation Module Specification" by Chris Lattner, December 1999 (hereinafter "Lattner"), claims 12, 29, 39 and 47 as being unpatentable over Wang in view of Lattner, and further in view of "The Principles of Computer Hardware, Third Edition" by Alan Clements, 2000 (hereinafter "Clements"), claims 20 and 22 as being unpatentable over Wang and Lattner, and further in view of "Load-Time Structural Reflection in Java" by Shigeru Chiba, June 2000 (hereinafter "Chiba").

Regarding claim 11, Applicant submits that Wang fails to teach generating a platform-independent representation of the one or more script language instructions. In response to Applicant's previous arguments, the Examiner states, in the Final Office Action, "the notify and wait method invocations as taught by Wang are equivalent representations of the script language instructions." Applicant respectfully disagrees with the Examiner's interpretation of Wang.

Wang teaches a system wherein multiple intermediate sources are created from an original source document to enable multiple processors to interpret and process their respective intermediate source. Wang teaches that when creating the intermediate source of a specific processor, blocks from the original source that are not appropriate for the specific processor are replaced with synchronization and/or notification tokens to allow synchronization between the processors (Wang, column 3, lines 31-49). Specifically, Wang teaches a server system 106 executing a Web daemon 108 including one or more runtime processors, which may comprise a Java Virtual Machine 110 and a VisualBasic

Script interpreter 112. The server system may further include one or more translators 114 that are operable to translate an original input source for the one or more runtime processors. (col. 2, lines 40 – 58) In one embodiment the translator 114 may translate the original input source into two intermediate sources, one for the Java Virtual Machine 110 and one for the VisualBasic Script interpreter 112. The translator may also translate every HTML block of the original source into a synchronizer token. Execution flow may then move back and forth between the Java Virtual Machine 110 and the VisualBasic Script interpreter 112 according to said tokens. (col. 3, line 25 – col. 4, line 30). Wang does this so that each processor has an input file containing only those instructions it needs to process plus the synchronization information necessary for ordering the overall processing of the original source. Thus, rather than teaching a system that generates a platform-independent representation of script language instructions, Wang teaches a system whereby multiple run-time processors may all work on the original, platform specific, script language instructions.

Wang also teaches that when creating the intermediate source for a specific processor, those blocks from the original source that are not appropriate for the specific processor are replaced with synchronization and/or notification tokens to allow synchronization between the processors (Wang, column 3, lines 31-49). Applicant submits that the notify method and wait method invocations as taught by Wang are clearly not equivalent representations of script language instructions. Wang specifically, and clearly, states that "the HTML Parser 114 translates the first VisualBasic Script *block* into a thread object run method invocation, a notify method invocation, and an immediate what method invocation, wherein the thread object run method initiates execution of the VisualBasic Script interpreter 112, the notify method sends a notification to the VisualBasic Script interpreter 112, and the immediate wait method waits for a notification to be received from the executing Visual Basic Script interpreter 112." (Wang, column 4, lines 48-58). Thus, the notify method and wait method invocations are not equivalent representations of script language instructions. They are simply synchronization calls to coordinate the various script language specific interpreters utilized in Wang's method.

Applicant further submits that at the above-cited passage, Wang is stating that a *block* of VisualBasic instructions is replaced with method calls that only invoke a VisualBasic Script Interpreter, notify the interpreter of the need to interpret VisualBasic script instructions, and wait for a notification from the interpreter. Wang further clearly states, "the HTML Parser 114 translates the remaining (i.e., second or later) VisualBasic Script *blocks* in the original input source 116 into notify method and wait method invocation." (parenthesis in original) (Wang, column 4, line 65-column 5, line 1). Thus, every *block of script language instructions* is replaced with an identical set of notify and wait method invocations regardless of the actual instructions in the block that is being replaced. Applicants can find no conceivable way that these notify method and wait method invocations can be interpreted as equivalent representations of the original script language instructions. Following the Examiner's logic (that the notify an wait method invocations as taught by Wang are equivalent representations of the script language instructions), every block of script language instructions are the equivalent to notify and wait method invocations. This is clearly an incorrect interpretation of Wang's teachings. Further, Applicant notes that claim 11 specifically recites, "wherein the platform-independent programming language representation comprises a sequence of logical commands representing the sequence of scripting language instructions." Applicant submits that Wang positively fails to teach such a system.

The Examiner further states, "Wang clearly discloses converting instructions into an equivalent form, and hence the instructions are not left unchanged", (Final Office Action, page 3, lines 21-22) and cites Wang, column 1, lines 19-21 for support. Applicant submits, however, that Wang, at the Examiner's cited passage, is disclosing precisely the opposite. Rather than generating a platform independent representation of script instructions, Wang, at the Examiner's cited passage is disclosing translating the entire input source into *equivalent script instructions*. Wang's example, which the Examiner also cites in support of his assertion, bears this out. It is an example, where HTML (a platform independent language) text is converted into VisualBasic Script, "so that a VisualBasic Script interpreter only has to interpret the VisualBasic Script and not

the HTML text." (Wang, column 1, lines 22-30). Thus, the Examiner's cited passage actually discloses translating non-script instructions into equivalent script instructions and does not disclose converting script instructions into an equivalent platform-independent representation, as recited in Applicant's claim 11.

In response to Applicant's previous argument that the Examiner is improperly combining both the teachings of Wang with the deficiencies of a system from which Wang is specifically teaching away, the Examiner states, "it is inherent to convert a script into an equivalent intermediate form" and further states, "[s]ince this aspect is inherent, and well known, it can used as a prior art." The Examiner points only to column 1, lines 19-21 for support. As shown above, the cited reference actually fails to teach the aspect that the Examiner claims is inherent. Specifically, the cited reference does not teach converting script instructions into an equivalent form, but instead teaches the *conversion of non-script portions of an input source into equivalent script instructions.* Thus, Applicant submits that it is not inherent to convert a script into an equivalent intermediate form, as the Examiner contends and therefore the Examiner has failed to provide prior art that teaches such conversion.

The Examiner also asserts, "converting a script into multiple intermediate sources does not negate the prior art teaching of converting a script into an equivalent form." Applicant submits that even if the cited reference did teach converting a script into an equivalent form, which, as Applicant has shown above, it does not, Applicant still submits that it is improper to combine the teachings of Wang with the deficiencies of a system from which Wang is specifically teaching away and which his system is specifically designed to counter. Applicant directs the Examiner to the first line of Wang's summary which states, "[t]o overcome the limitations in the prior art described above," specifically referring to the Examiner's cited reference. Applicant submits that since Wang specifically and directly teaches away from converting the non-script portions of an input source into an equivalent script instructions, a combination of Wang with such teaching would clearly render Wang's system unsatisfactory for its intended purpose. (See, M.P.E.P. § 2143.01, pp 11).

Regarding Applicants previous arguments that Lattner does not suggest representing script instructions as executable platform-independent programming object, the Examiner states, "since Lattner teaches the use of a Java-based Instruction class to represent 80x86 assembly instructions, this Instruction class can obviously be generalized to all type of instructions, include script instructions." Applicant respectfully disagrees with the Examiner. Lattner teaches an Instruction class which is designed to exist as a node in a linked list and generates a legal 80x86 assembly language string for an intermediate language instruction associated with that particular node in the list (page 2). 80x86 assembly language instructions are of a wholly distinct and different character than embedded script instructions. Applicant further disagrees with the Examiner assertion that "Lattner teaches the benefits of representing an Instruction as an object, and thus it would be beneficial to represent a script instruction as an object." Lattner teaches that the benefit of representing 80x86 instructions as objects is that it allows "80x86 specific optimizations to take place, such as instruction selection, register allocation, and peephole optimization in an efficient and elegant way." (Lattner, page 1, lines 33-36). Thus, Applicant submits that the benefits to representing 80x86 instructions as objects, as taught by Lattner, only refer to 80x86 specific optimizations, and do not provide any motivation to combine Lattner's teachings with the embedded script processing system of Wang. Applicant submits that there is no suggestion in the prior art to apply Lattner's Java-based Instruction class to represent anything other than 80x86 assembly instructions. Assembly instructions and script instructions are handled very differently in computer systems. Prior art teachings in regard to assembly instructions do not automatically extend to script instructions, as the Examiner suggests. Applicant submits that the Examiner is merely using hindsight analysis to infer the obviousness of extending Lattner's teachings to include script instructions.

Therefore, Applicant again submits that Wang and Lattner individually, as well as the combination of Wang in view of Lattner, fail to teach or suggest "generating a platform-independent programming representation of the sequence of script language instructions" and further fails to teach or suggest "wherein the platform-independent

programming language representation comprises a sequence of logical commands representing the sequence of script language instruction, and wherein each of the logical commands is stored as one or more platform-independent programming language objects" as the Examiner contends and as recited by the Applicant's claim 11. Further, Applicant submits that even if the combination of Wang in view of Lattner did teach such a system, while Applicant maintains that it does not, such a combination is improper since neither Wang nor Lattner provide any motivation or benefit to apply the object based 80x86 instruction representations as taught by Lattner with the script instruction processing of Wang.

In light of the above remarks, Applicant submits that the rejection of claim 11 is not supported by the cited art and withdrawal of the rejection is respectfully requested. Likewise, independent claims 28, 38 and 46 recite subject matter similar to Applicant's claim 11, and are thus believed to patentably distinguish over the cited art for at least the reason given above.

Regarding claim 12, in response to Applicant's previous argument that Clements does not teach using a stack structure to hold instructions, the Examiner responds that Clements "does teach in Section 6.5.2 the use of a program stack and program counter, which stores instructions on a stack." Applicant submits, however, that Section 6.5.2 of Clements teaches the use of a stack to store arguments and return addresses when making subroutine calls. Applicant further submits that the Examiner's cited passage (Clements, page 7, lines 3-4) specifically states, "the value of the PC pushed on stack during a subroutine call is usually the *return address* (i.e. the address of the next instruction following the subroutine call)" (parenthesis in original, emphasis added).

As noted previously (see Applicants response mailed April 8, 2004) Clements teaches, "a computer can *transfer data between memory and the stack*, and perform monadic operations on the top item of the stack, or dyadic operations on the top two items of the stack" (Emphasis added) (Clements, Section 6.5, paragraph 4). Clements further teaches the use of the stack to store the parameters and the return addresses of

subroutines (Clements, Sections 6.5.2 and 6.5.4). Thus, Clements does not teach, at the Examiner's cited passage or elsewhere, storing instructions on a stack.

Applicant further submits that the Examiner's assertion that "addresses of instructions are equivalent to the instructions themselves" is incorrect. Applicant notes that it is extremely well known that addresses, even addresses of instructions, are very different from, and thus not equivalent to, instructions themselves. Addresses are a means of referencing a particular portion of memory in a computer system. What resides at any particular portion of memory, and therefore at any particular address, may vary from time to time during the execution of a computer system. As programs, or portions of programs, are loaded and unloaded (or otherwise moved around in memory), the particular data or instruction that resides at a particular address may change. In other words, a single address may point to very different instructions at different times (or even to data as some times and instructions at other times). Hence, the address of an instruction is very different from the instruction itself.

As shown above, Wang and Lattner fail to teach the method of Claim 11. Further, Clements fails to teach using a stack structure to hold instructions. Therefore, Applicant submits that the combination of Wang in view of Lattner, in further view Clements fails to teach or suggest, "wherein the platform-independent programming language objects of the platform-independent programming language representation are stored in a stack data structure, and wherein said executing is performed by a stack-machine interpreter engine configured to pop the one or more platform-independent programming language objects for each of the logical commands of the platform-independent programming language representation off the stack data structure when executing the particular logical command of the platform-independent programming language representation" as recited by Applicant's claim 12, and as suggested by the Examiner.

Therefore, in light of the above remarks, Applicant asserts that the rejection of claim 12 is not supported by the cited art and withdrawal of the rejection is respectfully

requested. Similar remarks as discussed above in regard to claim 12 apply to claims 29, 39, and 47.

Regarding claim 21, Applicant again submits that the rejection is improper because the Examiner has failed to give any reason or motivation to combine removing methods and fields from a program object, which the Examiner asserts is well known, with the script interpreting system of Wang. Applicant respectfully reminds the Examiner that merely stating that individual aspects of a claimed invention are well known does not render the combination well known without some objective reason to combine the individual teachings. *Ex parte Levengood*, 28 USPQ2d 1300. As the Court of Appeals for the Federal Circuit recently explained in *In re Sang Su Lee*, Docket No. 00-1158 (Fed. Cir. January 18, 2002), conclusory statements such as those provided by the Examiner that a claim limitation is well known or common knowledge do not fulfill the Examiner's obligation. "Deficiencies of the cited references cannot be remedied by the [Examiner's] general conclusions about what is 'basic knowledge' or 'common sense.'" *In re Zurko*, 59 USPQ2d 1693, 1697 (Fed. Cir. 2001). **The Examiner did not provide any such reasoning and has failed to respond to this specific argument when previously presented** (see Applicant's response dated April 8, 2004). Thus the rejection of claim 21 is improper and its removal is respectfully requested. Similar remarks as discussed above in regard to claim 21 apply to claim 23.

Applicant also asserts that numerous ones of the other dependent claims recite further distinctions over the cited art. However, since the independent claims have been shown to be patentably distinct, a further discussion of the dependent claims is not necessary at this time.

CONCLUSION

Applicants submit the application is in condition for allowance, and an notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5181-64300/RCK.

Also enclosed herewith are the following items:

☒ Return Receipt Postcard

☐ Petition for Extension of Time

☐ Request for Approval of Drawing Changes

☐ Notice of Change of Address

☐ Fee Authorization Form authorizing a deposit account debit in the amount of $ for fees ( ).

☐ Other:

Respectfully submitted,

Robert C. Kowert
Reg. No. 39,255
ATTORNEY FOR APPLICANT(S)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: July 30, 2004